NextGen Java Developer

עם התמחות ב-Big Data

430 שעות אקדמיות

שפת Java מושכת ומעניינת יותר ויותר אנשים ולצד הביקוש הגבוה למפתחי Java טובים. לעומת השנים הקודמות, בעשור האחרון לימוד Java הפך לנגיש מאוד - הודות ספרים, מאמרים, הדרכות, קורסים בכל הרמות, קהילה גדולה של מפתחים, ומעל 12 מיליון מפתחי Java שמתמצאים בשפה ברמה גבוהה ומוכנים לתרום מהידע שלהם וגם מקבלים שפע של ידע בחזרה.

שפת Java החלה להיות בשימוש נרחב ביישומים עסקיים בתחילת שנות ה-2000. היא נוצרה מכמה סיבות עיקריות: **פשטות, עמידות וגמישות (ניידות)**. השפה הומצא לפני יותר מ-20 שנה, ובמשך שנים רבות זו השפה הפופולרית והמובילה ביותר לפיתוח צד שרת - backend development.

פשטות: בראש ובראשונה, Java מציעה למפתחים שפה פשוטה יותר מאשר ++C למשל. אכן, פונקציות מפתח רבות של ++C לא נוספו לשפת Java , אבל המטרה הייתה לפשט. ירושה מרובה - Multiple inheritance - היא דוגמה לכך.

עמידות: Java פותחה כשפה חזקה יותר. הרבה באגים שקשה לאתר בעבודה עם ++C, לא קיימים ב-Java . ע"י מתן ניהול זיכרון אוטומטי, (כמעט) אין דליפות זיכרון שהיו אז בשפות C, ++C.

גמישות: Java משמשת כשפה ניידת. היכולת לפתח, לקמפל ולבנות יישום על מכונת Windows ולפרוס אותו בשרת Linux הייתה מהפכה בפיתוח תוכנה.

Java שומרת על התכונה של תאימות לאורך שנים רבות, וזה עדיין תקף גם היום. היתרון הברור: חברה שהשקיעה בפיתוח ב- Java לפני 20 שנה עדיין נהנית מאותה השקעה. יישום שנכתב והוכן ב-Java בשנת 1998 באמצעות Java 2 עדיין עובד בסביבה Java 12 ללא צורך לבנות אותו מחדש. אין שפה אחרת או פלטפורמה שמציעה בטחון ברמה כזו.

במהלך השנים השפה עברה שינויים רבים, רכשה מספר עצום של ספריות (3rd party libraries), סביבות פיתוח - frameworks, כלים שונים, עקרונות חדשניים, מתודולוגיות ופרדיגמות.

היום, כדי לכתוב ב- Java ברמה מקצועית, זה לא מספיק רק ללמוד את הסינטקס של השפה ואת העקרונות הבסיסיים של תכנות מונחה עצמים. היום, Java היא עולם עצום של כלים וטכנולוגיות, Ecosystem שלם ולכן רק המתכנתים שצברו ניסיון רב בכלים אלה באמת מוערכים בתעשייה.

טענה זו מתחזקת עוד יותר עם התפתחות הטכנולוגית לכיוונים של DevOps, Big Data וכניסה לעולמות הענן.

תיאור התפקיד:

מפתח Java טוב מתאפיין במספר תכונות/יכולות:

- **תכנות מונחה עצמים – Object Oriented Programming** - מפתחי Java טובים מיומנים ביישום תבניות עיצוב מונחות עצמים - object oriented design patterns - ומסוגלים לעצב את הקוד באופן יעיל.
- **ידע בשימוש ומימוש כלים וטכנולוגיות רבות** - מפתחי Java טובים מיומנים בשימוש בטכנולוגיות מרובות, כגון: Spring, Spring Boot, Spring Cloud, Maven, Gradle, JUnit, Hibernate, ועוד. Reflection – עוד נושא שקריטי לארגז כלים של מפתח – להכיר לעומק את הבנייה של המערכת מבפנים.
- **מאגרי מידע** – יכולת לקרוא ולהבין את המורכבות של שאילתות SQL היא דבר נוסף שמפתחי Java צריכים להיות טובים בו. היכרות מעמיקה עם מושגים כמו joins, aggregations, indexing - הם חלק ניכר מיכולות המפתח.
- **Data Structures & Collections** - מפתחי Java צריכים לדעת לא רק מה זה List, Map, Set. נדרש ידע ב-Stream API.

- **JVM ו-Memory Management** - אלה הם כמה מן הנושאים מסובכים שמפתחי Java מיומנים בהם. הבנה טובה על זמני הריצה וכיצד נוצר garbage collection הוא ידע חיוני להצלחה שלהם.
- **Microservices Architecture** – מתודולוגיה של בניית מערכת. אם פעם כתבו תוכנה בפרויקט אחד גדול שכלל כלים ופתרונות רבים, היום הנטיה הינה לבנות מספר רב של פרויקטים קטנים יותר אשר מתקשרים אחד עם השני.
- **יכולות פיתוח בסביבת Big Data** – עם הכניסה לעולמות ה-Big Data, ושימוש אינטנסיבי בשפות בתהליכי עבודה על דאטה, נדרש ידע לבניית סביבה עבודה על נתונים באמצעות Spark.
- **עבודה בצוות** - זה אולי נראה ברור, אבל תקשורת טובה ויכולת עבודה בצוות הן בהחלט תכונות שמגדירות מפתח טוב. תקשורת בתוך הצוות ועם בעלי העניין היא חיונית.

תיאור הכשרה:

מסלול הכשרה זה יעזור לסטודנטים לא רק להכיר את הקוד ואת החידושים האחרונים בשפה בגרסאות האחרונות שלה, אלא גם יאפשר היכרות מעשית עם תחומים בהם Java תופסת מקום נכבד, כגון DevOps, Big Data, היכרות מעמיקה עם frameworks הפופולריים ביותר, הכלים ו-design patterns, והכי חשוב יספק לסטודנטים את ההזדמנות לרכישת ניסיון רב ערך באמצעות תרגול אינטנסיבי ופרויקטים שיבוצעו במהלך הקורס.

רוב המטלות מבוססות על מגוון פרויקטים מהתעשייה שעומדים בפני המדריכים שלנו מדי יום. בקורס אחד, הבוגרים ירכשו ידע פרקטי ומעשי במתודולוגיות המתקדמות ביותר, כולל DevOps, Big Data, Microservices.

בסוף הקורס, נעזור לכם לכתוב קורות חיים בצורה נכונה, ונשתף את הניסיון שלנו על הדרך הנכונה לעבור ראיון עבודה בחברות מודרניות אשר מחפשות אנשים עם ניסיון מתאים. כמו כן, מחלקת גיוס של נאיה טכנולוגיות תבחן אפשרות קליטה של בוגרים מצטיינים לפרויקטים שבה מעורבת אצל לקוחות רבים, לאחר שהבוגרים המצטיינים יעברו תהליך בחינה מעמיק.

קהל יעד:

מסלול הכשרה זה מיועד למפתחים בשפות שונות עם ידע ונסיון, המעוניינים לעשות הסבה לשפת Java. כמו כן בוגרי תארים אקדמיים בתחום הנדסת תוכנה עם ידע בשפות תכנות עיליות.

דרישות קדם:

- יעוץ וראיון אישי
- מעבר מבחן כניסה
- ידע ונסיון בפיתוח בשפות OOP
- היכרות עם מסדי נתונים ושפת SQL

תכני המסלול:

<u>Java SE and OOD</u>

- **Java Basics**
  - Interpreted language overview
  - Control statements
  - Primitives
  - Operations
  - Strings and Arrays
  - Objects and Classes
  - Static context
  - Method calls
  - Inheritance
  - Interfaces, abstract classes and methods
  - Enums, Inner Classes, Packages

- **IntelliJ IDEA**
  - Overview
  - Code completion
  - Life templates
  - Classes and objects operations and navigation
  - Refactoring
  - Plugins
  - Debugger
- **Concepts**
  - Object Oriented Design
  - Encapsulation principals
  - Casting and polymorphism
- **Java Basics**
  - Java API usage:
    - lang package
    - math package
    - util package
  - Memory management - GC, Runtime, System
  - Exceptions handling
  - JARing
  - JavaDocs
- **Testing**
  - JUnit basics
  - JUnit overview
  - Mockito
  - Power mock
- **Java Advanced**
  - Multithreading
  - Collections Framework
  - JDBC
  - Input/Output
  - Serialization
  - Reflection
  - Networking
- **Build Tools**
  - Maven basics
  - Gradle basics
  - Maven advanced
  - Gradle basics
- **Rolling-on Project**
  - Overview of Book-a-flight project
  - Flat-file local access command line implementation
  - Writing simple Maven build, guided
- **XML**
  - Overview

        - Tags and attributes
        - Schema
    - XML- API
        - JAX-B, eXtream, more
- **JSON**
    - Jackson
    - Gson
    - Serializers
- **Desktop UI**
    - Swing introduction
    - Java Events model
- **Rolling-on Project**
    - Writing Maven build file
    - Writing custom plugins
- **Performance**
    - JVM tuning
- **Source control tools**
    - GIT
    - Intellij plugin
- **Design patterns**
    - Anti-patterns
    - SOLID
    - Strategy
    - Singleton
    - Factory
    - Builder
    - Proxy
    - Decorator
    - Chain of responsibilities
    - Adapter
    - Command
    - Observer
- **Multithreading**
    - Threads
    - Runnable / Callable
    - How to stop threads correctly
    - Synchronization / volatile / locks
    - Wait / notify
    - Executors
    - ThreadLocal
    - Immutable objects
    - Reactive Java
- **Reflection, Java internals, 3-d party libraries**
    - Java optimization
    - ClassLoaders
    - Classpath

  - Class / Method / Constructor / Field
  - Annotations
  - org.reflections
  - How to scan packages
  - Dynamic proxy
  - Invocation handler
  - CGLIB
  - Frameworks
  - Mockups
  - ReflectionUtils
- **GC**
  - Java memory model
  - Heap / stack / perm (till Java 7)
  - Young / old generations
  - GC & memory parameters
  - G1
- **Profiling Monitoring & Instrumentation**
  - VisualVM
  - MBeans
  - VisualGC
  - Agents
- **Lombok**
  - Annotation Processors
  - Pojo annotations
  - Functional annotations
  - Delombok

JAVA 8

- **Lambda expression**
  - Motivation
  - Callback method
  - Anonymous inner classes
  - Closures
  - Lambda expression syntax
  - Functional Interface
  - Variable scopes
  - Method reference
  - Static method reference
  - Instance method reference
  - Functional Interfaces & Collections
  - Predicate
  - Function
  - Supplier
  - Consumer
  - forEach, replaceAll and other friends of Collections
  - Unary Operators
- **Multiple Inheritance in Java 8. Is it true?**
  - Method in interfaces
  - Default method

  - Differences between interfaces and abstract classes. Yes, again!
  - Static methods in interfaces
- **JSR 310 Data & Time like Joda and even better.**
  - Immutability
  - Date Classes
  - Time classes
  - Formatting Date and Time
  - Migrating from JDK 1 to JDK 8 ☺
- **Reflections – what changed?**
  - Local parameters
  - Repeatable annotations
- **Streams in Java 8**
  - Streams & collections
  - Specific Streams
  - Filters
  - Laziness & Parallelism
  - Collectors
  - Downstream collectors
  - Map reduce
  - Files Additions
  - Creating streams
    - Builders
    - Generators
    - Spliterator
  - Performance issues
- **Optional**
  - Manada concept
  - Functional style with optional
  - Patterns and anto-patterns

<u>Spring Framework</u>

- **Spring Core**
  - Design patterns
  - Factory
  - Singleton
  - Chain of responsibilities
  - Proxy
  - Annotations
  - Dynamic Proxy
  - CGLib
  - Spring concepts
  - EJB history
  - IOC
  - Dependency Injection
  - Container
  - Bean
  - Spring XML context
  - Bean lifecycle

- Init / destroy method
- BeanPostProcessors / BeanFactoryPostProcessors
- ApplicationListeners
- FactoryBeans
- Annotation Config
- Standard and Spring annotations
- Qualifiers
- Java Config
- Bean Declaration
- Static beans
- Refreshing prototypes in singletons
- Proxy Mode
- Java Config problems
- What is better (Java Config / Xml / Annotations)
- Spring profiles / Conditional
- Tests with Spring
- Spring AOP
- Aspect
- Advice
- JoinPoint
- @Before / @After
- @AfterReturning / @AfterThrowing
- @Around
- Pointcut
- Weaving
- Best practice
- Custom annotations
- Exception handling
- Logging

- **Spring Data**
  - Main concept
  - Repository interface
  - Dynamic proxy
  - Custom methods
  - Annotations

<u>Hibernate & JPA</u>

- Hibernate general overview
- Defining persistent Java classes
- Mapping POJO classes to DB
- Identity generators
- Mapping inheritance hierarchy
- DB connectivity
- Configuration
- The SessionFactory
- Session
- Object identity
- Identity types
- Surrogate ids

- Equals and hash code
- Persistent object lifecycle
- Lazy initialization of collections
- Entity association mapping
- 1-1 / 1-n / n-m
- Cascade
- Using JPA
- The Hibernate Entity Manager
- JPA Annotations
- Hibernate JPA extensions
- Querying
- Transactions
- The 2nd level cache
- Spring-ORM
- EntityManagerFactoryBean
- Persistent context
- Transaction Manager
- Declarative transactions
- Isolation
- Propagation
- Nested Transactions
- Proxy problems
- SelfInject solution

Microservices

- **Spring Boot**
  - How magic happens
  - @Conditional
  - @OnBeanCondition
  - Spring-boot-maven-plugin
  - start.spring.io
- **Web & Spring MVC**
  - Servlets and JSP
  - Rest
  - Controllers / RestControllers
  - RestTemplate
  - Tomcat
  - Spring test
- **Spring cloud**
  - Discovery Service
    - Eurika Server
    - Eurika Client
    - Two approaches
    - Zul and load balancer
    - Qualifiers when working with several RestTemplates
    - Config Server
  - Common properties
    - Specific properties
    - Reloadable properties

    - ScopeRefresh

- **Spring additional modules**
    - Spring Data Rest
    - Spring Integration
    - Spring Batch
    - Spring Jdbc Template
    - Spring Social
    - Spring Data Mongo
    - Spring Security

Devops

- **Continuous Integration with Apache Maven, Jenkins-CI and Artifactory**
    - Introduction to Continuous Integration
    - Apache Maven 3
    - Jenkins-CI
    - Build Tools Landscape
    - Maven & Procedural Build Tools
    - The POM
    - The Build Lifecycle
    - Standard Project Layout
    - Running Maven
    - Artifacts & Dependency Management
    - Bom
    - Repositories
    - POM Inheritance
    - Cross-project Configuration
    - Profiles
    - Installation and Deployment
    - Plugins
    - Lifecycle and Packaging
    - Version Control
    - Archetypes
    - Site and Project Reports
- **Jenkins**
    - Configuration
    - Jobs
    - Popular plugins
- **JFrog Artifactory**
    - Introduction
    - Installation and Setup
    - Repositories
    - Security
    - Using the UI
    - Working with Maven
    - Working Jenkins
    - General Configuration

Data Engineering

- **Introduction to Big Data**
  - Data locality
  - Map reduce
  - New approach
  - Hadoop implementations
- **Hadoop ecosystem**
  - HDFS
  - HBase
  - YARN
  - Hue
  - Sqoop
  - Flume
  - Hive
  - Impala
  - Oozie
- **Introduction to SCALA**
  - Singleton objects
  - Functional programming
  - Tuples
  - Matcher pattern
- **Spark**
  - Spark intro
  - Running spark in cluster (spark submit)
  - Spark API
  - RDD
    - Transformations
    - Actions
    - Creation
    - Testing
    - File types (SCV, parquet)
  - Comparing RDD API (Scala vs Java)
  - Partitioning
  - Broadcast and accumulators
  - Developing @Broadcast annotation
  - DataFrames API
    - SqlContext
    - Working with schema files
    - Json, avro
    - DataFrame methods
    - functions API
    - writing etl process
    - creating schema
    - creating dataframe from rdd and from SCV files
    - sql usage on dataframes
    - developing custom annotations with Spring for dataframes
  - Spark and Junit

- Writing tests for spark with spring runner
- Separating production from test environment with spring @Conditional
- Full stack application (from spark to rest service)
- Migration to Spark 2
    - Datasets
    - Spark streaming new API
- Working with Kafka
    - Queues brokers topics and listeners
    - Working with Kafka
    - Spark Streaming from Kafka